

Tehnici Web

CURSUL 9

Semestrul I, 2024-2025
Carmen Chirita

XML - Extensible Markup Language

- este un limbaj de marcare similar cu HTML
- nu conține taguri predefinite, utilizatorul își definește propriile taguri și structura documentului
- este conceput pentru stocarea și transmiterea datelor
- este independent de software și hardware
- poate fi utilizat pentru a crea limbaje de marcare precum XHTML, SVG, MathML, etc.

Structura unui document XML

Un document XML este format din:

- elemente (taguri: <nume_tag>; sunt case sensitive)
- date caracter (continutul elementelor)

Referințe de entitate predefinite în XML

<	<	less than
>	>	greater than
&	&	ampersand
'	'	apostrophe
"	"	quotation mark

Sursa imaginii: https://www.w3schools.com/xml/xml_syntax.asp

Structura unui document XML

- se definește versiunea XML și codarea caracterelor (prima linie în document)

```
<?xml version="1.0" encoding="UTF-8"?>
```

- toate documentele XML trebuie sa conțină un element rădăcina

```
<root>  
  <child>  
    <subchild>.....</subchild>  
  </child>  
</root>
```

document XML - structura arborescenta

Reguli de sintaxa XML

- exista un singur element rădăcina (root) într-un document XML
- fiecare element (tag) trebuie sa aibă tag de închidere
- elementele XML trebuie să fie corect imbricate
- attributele asociate elementelor nu pot conține mai multe valori

Exemplu-document XML

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore> //elementul radacina
  <book category="cooking"> //element copil
    <title lang="en">Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year>
    <price>30.00</price>
  </book>
  <book category="children">
    <title lang="en">Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
</bookstore>
```

Diagram illustrating XML structure and annotations:

- `//elementul radacina` points to the root element `<bookstore>`.
- `atribut` points to the `category` attribute of the `<book>` element.
- `//element copil` points to the child element `<book category="cooking">`.

XML-Parser

- browserele au un analizor XML incorporat pentru a accesa și manipula documente XML
- Înainte ca un document XML să poată fi accesat, acesta trebuie convertit într-un obiect XML DOM
- XML DOM definește proprietăți și metode pentru accesarea și editarea XML

Proprietăți XML DOM

nodeName

nodeValue

parentNode

childNodes

attributes

Metode XML DOM

getElementsByTagName()

appendChild()

removeChild()

XML-Parser

```
text = "<bookstore><book>" + "<title>Everyday Italian</title>" +  
"<author>Giada De Laurentiis</author>" + "<year>2005</year>" +  
"</book></bookstore>";    //XML ca string
```

```
parser = new DOMParser(); //se creaza un analizor XML DOM
```

```
xmlDoc = parser.parseFromString(text,"text/xml"); //se creaza un obiect XML  
DOM din stringul text
```

```
xmlDoc.getElementsByTagName("title")[0].childNodes[0].nodeValue;
```

```
//extragem informația din nodurile XML DOM
```


JSON = JavaScript Object Notation

<http://www.json.org/>

Ofera o modalitate de reprezentare a datelor, ca alternativa la XML.
Bazat pe JavaScript, este in prezent un format independent de limbaj.
Multe limbaje pot prelucra date in format JSON.
Este folosit pentru schimbul de informații cu serverul.

Elemente de baza:

<i>Object:</i>	<code>{"cheie1":val1, "cheie2":val2}</code>
<i>Array:</i>	<code>[val1, val2, val3]</code>
<i>Value:</i>	string, number, object, array, true, false, null

date.json

<pre>[{"pers": {"nume": "Ion", "varsta": 42} }, {"pers": {"nume": "Maria", "varsta": 30} }]</pre>

Valoare nu poate fi

function
date
undefined

Sintaxa JSON

Câmpul **cheie** trebuie să fie scris cu ghilimele

```
"nume": "Ana"
```

Câmpul **valoare** poate fi:

string, number, obiect (JSON), array, boolean, null

Obiectele JSON sunt reprezentate între acolade

```
{"nume": "Ana", "varsta": 30, "porecla": null }
```

Elementele array sunt reprezentate între paranteze drepte

```
[ "Ana", "Mihai", "Maria" ]
```

Valoare: string, number, object, array, true, false, null

JSON String: { "nume": "Andrei" }

JSON Number: { "varsta": 30 }

JSON Object: { "pers": { "nume": "Ion", "varsta": 42 } }

JSON Array: { "studenti": ["Ionut", "Mihai", "Dana"] }

JSON Boolean: { "promovat": true }

JSON null: { "porecla": null }

Obiectul JSON in JavaScript

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON

```
JSON.stringify(valoare) // transforma un obiect JavaScript intr-un string JSON  
JSON.parse(text)       //transforma un string JSON într-un obiect JavaScript
```

Exemplu:

```
var o1 = {pers: {nume:"Ion", varsta:42}},  
    o2 = {pers: {nume:"Maria", varsta:30}},  
    o = [o1,o2];  
  
var s = JSON.stringify(o);  
// '[{"pers":{"nume":"Ion","varsta":42}},{"pers":{"nume":"Maria","varsta":30}}]'  
  
localStorage.setItem("myarray", s);  
var st = localStorage.getItem("myarray");  
  
var jo = JSON.parse(st);
```

Poate fi folosit pentru memorare
in localStorage si sessionStorage

Obiecte JSON

```
myJSON = '{"cheie1":val1, "cheie2":val2, "cheie3":val3 }';  
myObj = JSON.parse(myJSON);
```

Accesarea obiectelor: `myObj.cheie1` sau `myObj["cheie1"]`

Iterarea proprietatilor unui obiect

```
<script>  
let myJSON = '{ "student":"Popescu", "grupa":231, "promovat":true }';  
let myObj = JSON.parse(myJSON);  
for (x in myObj) {  
    document.getElementById("prop").innerHTML += x + "<br>";  
}  
</script>
```

```
.....  
<p id="prop">
```

Paragraful va contine

student
grupa
promovat

Obiecte JSON

Iterarea valorilor proprietatilor unui obiect

```
<script>
let myJSON = '{ "student":"Popescu", "grupa":231, "promovat":true }';
let myObj = JSON.parse(myJSON);
for (x in myObj) {
    document.getElementById("val").innerHTML += myObj[x]+ " ";
}
</script>

.....
<p id="val"></p>
```

Paragraful va contine

Popescu 231 true

Obiecte JSON încorporate

```
let myJSON = '{ "student":"Ionescu",  
                "grupa":30,  
                "note": {"nota1":8,"nota2":9, "nota3":10}  
                }';  
let myObj = JSON.parse(myJSON);
```

Accesarea obiectelor încorporate:

```
myObj.note.nota2 // 9  
myObj.note["nota2"] // 9
```

Modificarea valorilor: `myObj.note.nota1="10";`

Stergerea proprietatilor: `delete myObj.note.nota1;`

JSON Arrays

```
myJSON = '[val1, val2, ..., valn]';  
myArray = JSON.parse(myJSON);
```

val1,...,valn pot fi string, number, object, array, boolean or null.

```
let dateJSON = '[{ "name":"Ford", "models":[ "Fiesta", "Focus", "Mustang"]},  
                  {"name":"BMW", "models":[ "320", "X3", "X5" ] },  
                  {"name":"Fiat", "models":[ "500", "Panda" ]}]';  
  
let myCars = JSON.parse(dateJSON) ;
```


Exemplu (array incorporat în array)

```
<head>
<script>
window.onload = function(){
  let x = "";
  let dateJSON = '[{"name":"Ford", "models": [ "Fiesta", "Focus", "Mustang"]},
                  {"name":"BMW", "models": [ "320", "X3", "X5" ] },
                  {"name":"Fiat", "models": [ "500", "Panda" ]}]';
  let myCars = JSON.parse(dateJSON) ;
  for (c of myCars) {
    x += "<h2>" + c.name + "</h2>";
    for (m of c.models) {
      x += m + "<br>";
    }
  }
  document.getElementById("demo").innerHTML = x;
}
</head>
<body>
  <p id = "demo"></p>
</body>
```

Ford

Fiesta
Focus
Mustang

BMW

320
X3
X5

Fiat

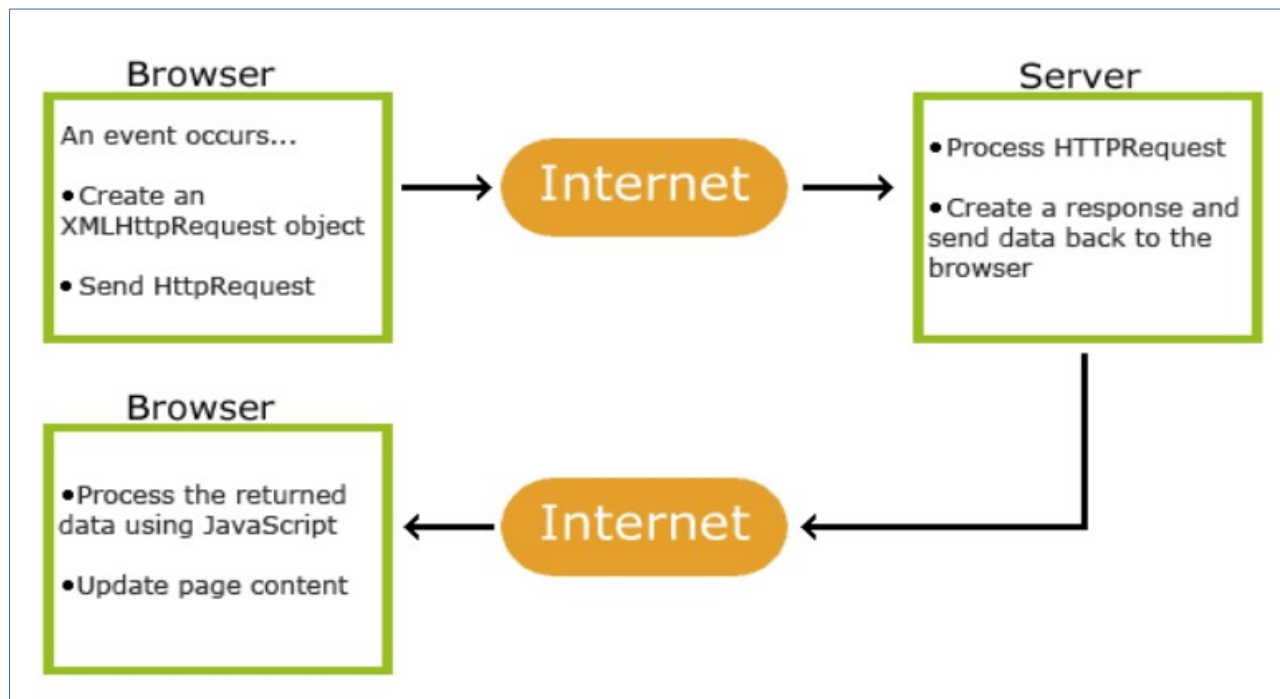
500
Panda

AJAX

- termenul „AJAX” a fost introdus de catre Jesse James Garrett in februarie 2005
- azi se refera la un grup de tehnologii folosite pentru procesele client ce aduc informație de la server fără reîncărcarea completă a unei pagini web
- initial se foloseau documente XML pentru request-uri și response-uri
- acum formatul JSON e mult mai uzual

Ajax

- permite actualizarea unor parti ale unei pagini web fără reîncarcarea completă a paginii
- trimite cereri către un server web și citește datele primite de la server
- nucleul sau îl reprezintă obiectul [XMLHttpRequest](#) care este folosit pentru a schimba date asincron cu serverul web



XMLHttpRequest

- XMLHttpRequest este un obiect JavaScript ce permite trimiterea de cereri către un server și returnarea rezultatului înapoi în script
- in plus, pot fi procesate in paralel mai multe conexiuni cu serverul, fara a bloca browser-ul pana la primirea raspunsului
- inainte de a putea utiliza XMLHttpRequest, trebuie creata o instanta a acestui obiect:

```
var xhr = new XMLHttpRequest()
```

("xhr" poate fi orice nume de variabila)

- pune la dispoziție mai multe proprietăți și metode pentru comunicarea client/server

XMLHttpRequest

onreadystatechange
//functia care se executa
la schimbarea starii cererii

readyState
// starea cererii
(0,1,2,3,4)

status
// codul de stare HTTP
200 pt OK

responseText
//raspunsul primit de la server în format text

responseXML
//raspunsul primit de la server în format XML

XMLHttpRequest

open()
//creaza cererea

send()
//trimite cererea

<https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest>

<http://www.javascriptkit.com/jsref/ajax.shtml>

Crearea unei cereri HTTP: metodele `open()` și `send()`

```
open( method, url, async ) // specifică tipul de cerere
```

`method`: poate fi GET sau POST

`url`: adresa serverului

`async`: true (asynchronous) sau false (synchronous)

```
send() // trimite cererea către server (se folosește cu GET )
```

```
send( date ) // trimite cererea către server (se folosește cu POST)
```

```
var xhr = new XMLHttpRequest();
```

```
xhr.open("GET", "http://localhost:3000/test.txt", true);
```

```
xhr.send();
```

Gestionarea raspunsului de la server

Proprietatea **readyState** reprezintă starea XMLHttpRequest

0: cererea este neinitializata, 1: conexiune stabilită cu serverul,
2: cererea a fost primita, 3: se proceseaza cererea,
4: cererea este finalizata și răspunsul este gata

Proprietatea **onreadystatechange** definește o funcție care trebuie executată când se schimbă readyState.

```
xhr.onreadystatechange = nume-functie;
```

Proprietatile:

status: codul de stare HTTP al raspunsului de la server, in format numeric (200 pt. "OK", 403 pt. "Interzis", 404 pt. "Negasit",etc)

statusText: statusul în format text ("OK", "Not Found")

Gestionarea raspunsului de la server

Când readyState este 4 și status este 200, răspunsul este pregătit

Accesarea datelor primite de la server

Proprietatea **responseText**: returneaza raspunsul primit de la server, in format text (string).

Proprietatea **responseXML**: returneaza raspunsul primit de la server in format XML.


Exemplu mdn

```
<script>
window.onload=function() {
  var httpRequest;
  document.getElementById("ajaxButton").addEventListener('click', makeRequest);

  function makeRequest() {
    httpRequest = new XMLHttpRequest(); //creaza un obiect XMLHttpRequest

    if (!httpRequest) {
      alert('Giving up :( Cannot create an XMLHTTP instance');
      return false;
    }
    httpRequest.onreadystatechange =alertContents;
    httpRequest.open('GET', 'http://localhost:3000/test.html');
    httpRequest.send();
  }

  function alertContents() {
    if (httpRequest.readyState === 4) {
      if (httpRequest.status === 200) {
        alert(httpRequest.responseText); //continutul fis. test.html
      } else {
        alert('There was a problem with the request.');
```



The screenshot shows an alert dialog box with a white background and a gray border. Inside the dialog, the following HTML code is displayed: `<!DOCTYPE html>`, `<html lang="ro">`, `<head>`, `<meta charset="utf-8">`, `<title>Exemplu Ajax</title>`, `<head>`, `</head>`, `<body>`, `Acesta este un test.`, `</body>`, and `</html>`. At the bottom right of the dialog is a button labeled "Ok".

Ok

Fișierul test.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<persoane>
  <pers nume="Ion" varsta = "40"></pers>
  <pers nume="Maria" varsta ="30"></pers>
</persoane>
```

Obiectul **XML DOM** in JavaScript
poate fi parcurs cu metode asemanatoare celor din DOM

```
var xml = httpRequest.responseXML;
var vpers= xml.getElementsByTagName('pers');
alert(vpers[0].getAttribute('nume'));
```

Exemplu XMLHttpRequest si XML

```
<script>
.....
httpRequest.open('GET','http://localhost:3000/test.xml');
httpRequest.onreadystatechange = alertContents;
httpRequest.send();
}
function alertContents() {
    if (httpRequest.readyState === 4) {
        if (httpRequest.status === 200) {

            var xmlDoc = httpRequest.responseXML;
            var per = xmlDoc.getElementsByTagName('pers');
            var content = "";
            for (var i=0; i < per.length; i++)
                content = content + per[i].getAttribute("nume") + " are " + per[i].getAttribute("varsta")
            + " ani \n";
            alert(content);

            .....
        }
    }
}
</script>
```

test.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<persoane>
  <pers nume="Ion" varsta = "40"></pers>
  <pers nume="Maria" varsta ="30"></pers>
</persoane>
```

Ion are 40 ani
Maria are 30 ani

Ok

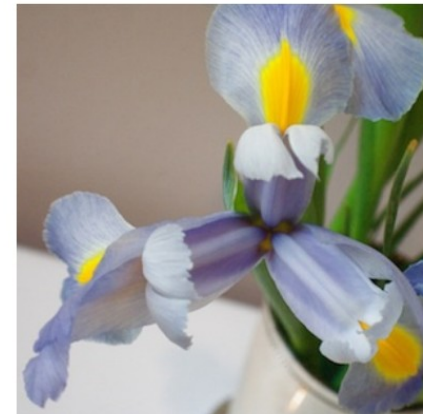
Exemplu: XMLHttpRequest si JSON

```
[  
  {"picture": {"caption": "Picture 1", "source": "images/flower1-300.jpg"} },  
  {"picture": {"caption": "Picture 2", "source": "images/flower2-300.jpg"} },  
  {"picture": {"caption": "Picture 3", "source": "images/flower3-300.jpg"} },  
  {"picture": {"caption": "Picture 4", "source": "images/flower4-300.jpg"} },  
  {"picture": {"caption": "Picture 5", "source": "images/flower5-300.jpg"} }  
]
```

pictures.json

```
var data ;  
var httpObj = new XMLHttpRequest();  
httpObj.open('GET', "http://localhost:3000/  
pictures.json", true);  
  
httpObj.onreadystatechange = function() {  
  if (httpObj.readyState == 4) {  
    if (httpObj.status == 200)  
    {  
      data=JSON.parse(httpObj.responseText);  
      gallery(data);  
    }  
    else {alert("eroare");}  
  }  
  
  httpObj.send(null);  
};
```

Gallery 1



Picture 1

Picture 1

Picture 2

Picture 3

Picture 4

Picture 5

```
function gallery(data){
  var menu_links = document.querySelectorAll("#menu button");
  var img_gal =document.getElementById("gal");
  var img_cap =document.getElementById("caption");
  for (var i=0; i< menu_links.length; i++)
  {
    let j=i;

    menu_links[j].onclick = function () {

      img_gal.src=data[j].picture.source;

      img_cap.textContent=data[j].picture.caption;

    }

  }
}
```

```
<body>
<h1>Gallery 1</h1>
<div id="container">
  <p>
    
  </p>
  <p id="caption">
    Picture 1
  </p>
</div>
<p id="menu">
  <button type="button">Picture 1</button>
  <button type="button">Picture 2</button>
  <button type="button">Picture 3</button>
  <button type="button">Picture 4</button>
  <button type="button">Picture 5</button>
</p>
</body>
```

Submiterea formelor

HTML

```
<body>
<form id="testform" method="post" action="http://localhost:3000/post">

<label>Nume:</label>
  <input type="text" id="nume" name="nume">
<label> Varsta:</label>
<input type="text" id="varsta" name="age">
<label>Localitate:</label>
<select name="city" id="loc">
  <option value="Bucuresti">Bucuresti</option>
  <option value="Timisoara" selected>Timisoara</option>
</select>
<button type="submit" id="buton"> Trimite </button>
</form>
</body>
```

Aplicația server

```
const express = require('express'); //folosim modulul express
const app = express(); //am creat serverul

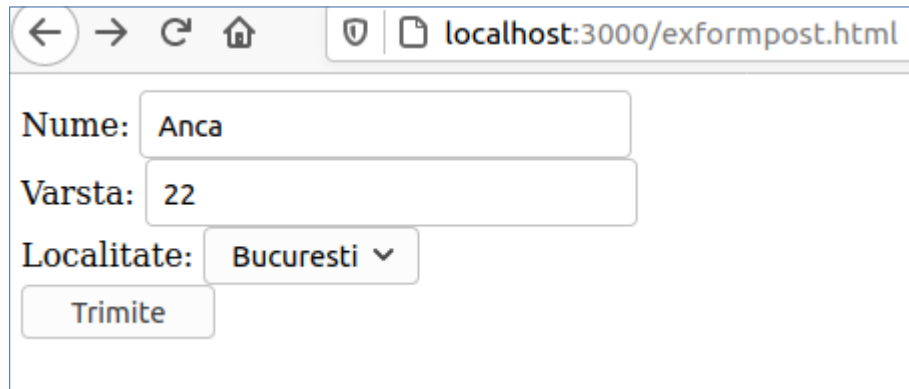
app.use(express.static('public')); //serverul intoarce
resursele statice din directorul "public"

app.use('/post', express.urlencoded({extended:true})); //
parseaza body-ul pentru formulare submise cu metoda post

app.post('/post', function(req, res) {res.send(req.body.name +
' din '
+ req.body.city + ' are ' + req.body.age + ' (de) ani'}})

app.listen(3000, function() {console.log('serverul asculta pe
portul 3000')}}) //pornirea serverului pe portul 3000
```

Submiterea formelor



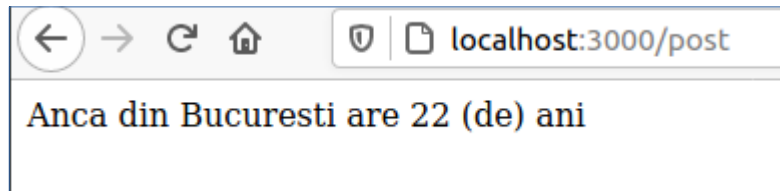
A screenshot of a web browser window. The address bar shows the URL `localhost:3000/exformpost.html`. The page contains a form with three input fields: "Nume:" with the value "Anca", "Varsta:" with the value "22", and "Localitate:" with a dropdown menu showing "Bucuresti". Below the form is a button labeled "Trimite".

Nume: Anca

Varsta: 22

Localitate: Bucuresti ▼

Trimite



A screenshot of a web browser window. The address bar shows the URL `localhost:3000/post`. The page displays the result of the form submission: "Anca din Bucuresti are 22 (de) ani".

Anca din Bucuresti are 22 (de) ani

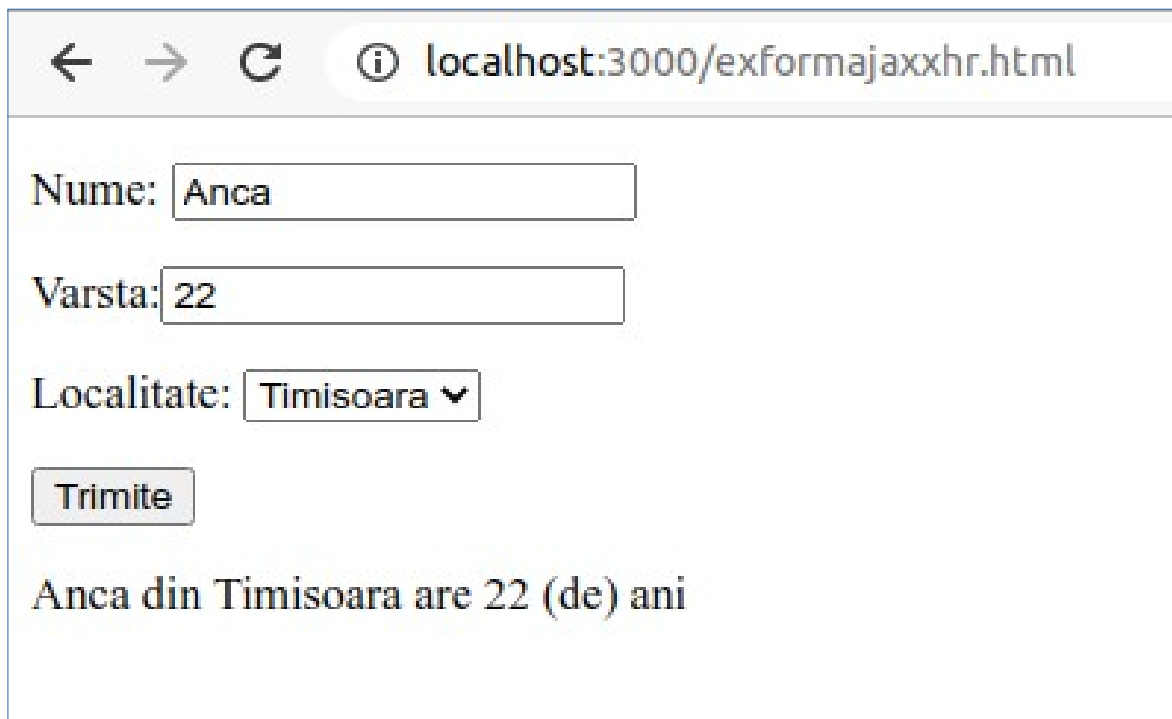
Submiterea formelor cu Ajax folosind XMLHttpRequest

JavaScript

```
window.onload=function () {  
    var forma=document.getElementById("testform");  
    forma.onsubmit= function (event){  
        event.preventDefault();  
  
        var dateleDinFormular = {nume:document.getElementById("nume").value,  
                                age:document.getElementById("varsta").value,  
                                city:document.getElementById("loc").value};  
  
        var xhr = new XMLHttpRequest();  
        xhr.open('POST', forma.action, true);  
        xhr.onreadystatechange = function(){  
            if (xhr.readyState == 4){  
                if (xhr.status == 200)  
                    {document.getElementById("info").innerHTML=xhr.responseText;  
                     else alert("error");  
                     }  
            }  
        }  
        xhr.setRequestHeader("Content-Type", "application/json")  
        xhr.send(JSON.stringify(dateleDinFormular));  
    }  
}
```


Aplicația sever

```
.....  
app.use('/post', express.json()); //parseaza datele din form (json) trimise cu post  
app.post('/post',function(req, res) {  
    console.log(req.body);  
    res.send(req.body.num + ' din '  
        + req.body.city + ' are ' + req.body.age + ' (de) ani');  
})  
.....
```



A screenshot of a web browser window. The address bar shows 'localhost:3000/exformajaxxhr.html'. The page contains a form with three input fields: 'Nume:' with the value 'Anca', 'Varsta:' with the value '22', and 'Localitate:' with a dropdown menu showing 'Timisoara'. Below the form is a button labeled 'Trimite'. Below the button, the text 'Anca din Timisoara are 22 (de) ani' is displayed.

← → ↻ ⓘ localhost:3000/exformajaxxhr.html

Nume:

Varsta:

Localitate:

Anca din Timisoara are 22 (de) ani

Promisiuni (promises)

„Cod producător”: de obicei necesita timp și întoarce un rezultat (ex: citește date dintr-o baza de date)

„Cod consumator”: dorește rezultatul „codului producător” odată ce este gata

- O promisiune este un obiect JavaScript care leagă „codul producător” și „codul consumator”
- Promisiunea pune rezultatul la dispoziția altor funcții („funcții consumatoare”)

<https://javascript.info/promise-basics>

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise

<http://www.javascriptkit.com/javatutors/javascriptpromises.shtml>

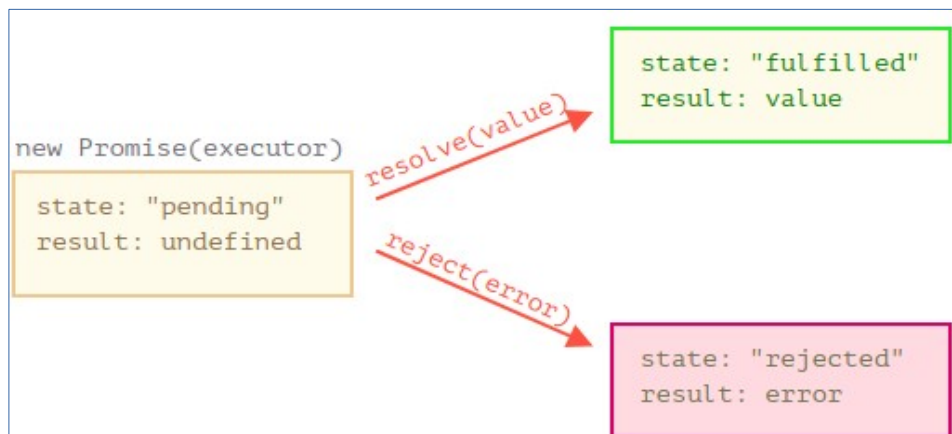
Promisiuni

Crearea unei promisiuni

```
promise = new Promise( function(resolve, reject) {  
    // cod producator care se executa asincron  
    resolve( val) // pentru succes  
    reject(err)   // pentru eroare  
} );
```

setInterval, setTimeout
cereri Ajax
evenimente

- la crearea unei promisiuni, se va apela automat functia transmisa ca argument promisiunii (numita funcție „executor”)
- „executorul” are ca parametrii doua functii predefinite (resolve și reject) și conține „codul producător” care ar trebui sa produca în cele din urma rezultatul
- în funcție de rezultatul obținut (codul asincron va decide dacă e succes sau eșec) se va apela una din cele doua funcții, functia de succes (resolve) sau functia de eșec (reject).



Starile obiectului promisiune:

în așteptare
rezolvată
respinsă

Promisiuni

```
// cod consumator care asteapta rezultatul  
promise.then(function (val) { //cod pentru succes},  
              function(err){ //cod pentru eroare})  
            .catch(function(err) { //tratarea erorii})
```

- „funcțiile consumatoare” pot fi înregistrate folosind metodele .then, .catch și .final.
- primul argument al lui .then este o funcție care rulează atunci când promisiunea este rezolvată și primește rezultatul
- al doilea argument al lui .then este o funcție care rulează atunci când promisiunea este respinsă și primește eroarea

Exemplu

```
let p= new Promise(function(resolve, reject) { //se creeaza o promisiune
    setTimeout(function(){
        var nota=Math.floor(Math.random() * 10) + 1);
        if(nota>=5)
            resolve("Bravo, ai promovat cu nota " + nota+ "!"); //apelam functia de succes
        else
            reject(nota); //apelam functia de esec

    },2000);});
console.log("Astept raspunsul");
p.then(succes, esec).catch(function(err){console.log(err)}); //daca primim o eroare
//o afisam

};

function succes(rez){//functia de succes
    console.log(rez);
}
function esec(err){//functia de esec

    throw new Error("Ai picat cu nota " + err + "!"); //aruncam o eroare care va fi prinsa de
//catch-ul promisiunii

}
```

Fetch

Fetch este o interfață pentru efectuarea unei cereri Ajax în JavaScript (alternativă modernă la XMLHttpRequest)

Metoda fetch():

`let promise = fetch(url, [options])` //intoarce o promisiune folosită pentru a obtine răspunsul de la sever

url: adresa url a severului

options: method, headers, body, etc. (parametrii optionalii)

https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Client-side_web_APIs/Fetching_data

<https://javascript.info/fetch>

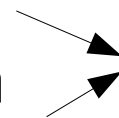
Fetch

```
let promise = fetch(url, [options]);  
promise.then(function(response){  
  return response.text();  
});
```

- promisiunea returnata de fetch este **rezolvata** cu un obiect din clasa Response imediat ce serverul răspunde cu antete
- promisiunea este **respinsa** dacă cererea nu poate fi trimisa (exista probleme de conexiune) sau nu se primeste răspunsul
- pentru a obtine răspunsul se folosesc metode suplimentare:

response.text() - răspuns în format text

response.json() - răspuns în format json



returneaza la rândul lor o
promisiune

Cerere Ajax folosind XMLHttpRequest

```
var data ;
var httpObj = new XMLHttpRequest();
httpObj.open('GET', "http://localhost:3000/
pictures.json", true);

httpObj.onreadystatechange = function() {
  if (httpObj.readyState == 4) {
    if (httpObj.status == 200)
    {
      data=JSON.parse(httpObj.responseText);
      gallery(data);
    }
    else {alert("eroare");}
  }
  httpObj.send(null);
};
```

Cerere Ajax folosind Fetch

```
var request = fetch("http://localhost:3000/
pictures.json");

request.then(function(response){
  if(response.status=='200')
    return response.text();
  else
    throw "eroare";
})
.then(function(text) {
  gallery(JSON.parse(text));
})
.catch(function(err){
  console.log(err);
});
```


Submiterea formelor cu Ajax (metoda POST) folosind Fetch

Codul JavaScript

```
window.onload=function () {  
  var forma=document.getElementById("testform");  
  forma.onsubmit= function (event){  
    var date = {nume:document.getElementById("nume").value,  
age:document.getElementById("varsta").value,city:document.getElementById("loc").value};  
    event.preventDefault();  
    fetch(forma.action,  
    {  
      method:"post",  
      headers: {'Content-Type': 'application/json'},  
      body: JSON.stringify(date)  
    })  
    .then(function(response) {  
      return response.text();})  
    .then(function(text) {  
      document.getElementById("info").innerHTML = text;  
    });  
  }  
};
```

Codul HTML

```
<form id="testform" method="post"  
action="http://localhost:3000/post">  
  <p> <label>Nume:</label> <input id="nume" type="text"  
name="nume"> </p>  
  <p> <label> Varsta:</label><input id="varsta" type="text"  
name="age"></p>  
  <p> <label>Localitate:</label> <select id="loc" name="city"></p>  
    <option value="Bucuresti" >Bucuresti</option>  
    <option value="Timisoara" selected>Timisoara</option>  
  </select>  
  <p><button type="submit" id="buton"> Trimite </button> </p>  
</form>  
<article id="info"></article>
```

Aplicația server

```
.....  
app.use('/post', express.json());  
app.post('/post',function(req, res) {  
  res.send(req.body.nume + ' din '  
    + req.body.city + ' are ' + req.body.age + ' (de) ani');  
})  
.....
```

Nume:

Varsta:

Localitate:

Maria din Timisoara are 19 (de) ani